B. AMENDMENTS TO THE SPECIFICATION

Please replace the paragraph at lines 17- 29 on page 2 with the following:

However, the above-mentioned methodologies are only capable of handling processor allocation in simple multiprocessor configurations, where various processing elements are non-differentiable in their functionality. With increasing application complexity and performance constraints, there has come up a need for different types of processors that can perform specialized functions, and can be completely dedicated for performing certain specific computations only. The current state of the art offers, in addition to general-purpose processors, multiprocessor systems having special-purpose processing elements. These special-purpose processors have access to a limited amount of private storage area, also referred as local program store, for the instructions that would be executed on these processors. These processing elements can be classified according to the types of computations they are capable of performing. Examples include digital signal processing (DSP) ~~DSP~~ processors, direct memory access (DMA) ~~DMA~~ engines, graphics processors, network processors and the like.

Please replace the paragraph at lines 15- 29 on page 8 with the following:

Referring now primarily to Fig.3, the basic steps of processor allocation process in multiprocessor system 100 are hereinafter described. At step 302, application program 116 ~~112~~, written as a series of interacting threads 118 ~~114~~, is loaded on compilation service 108 for compilation. The individual threads are then allocated to different processors as per the processing request, by processor allocation service 110. Certain threads do not require any specific processes to be performed on one of the special-purpose processors. Such a thread is allocated one of the free general-purpose processors 102 at step 304. This allocation can be done using a first-in-first-out (FIFO) strategy wherein a free processor receives the first thread request from the request-

queue. It would be evident to one skilled in the art that more complex strategies could also be used for processor allocation based on knowledge of parameters like task execution time, task priority, task pending time and task dependency. Examples include priority based preemptive scheduling (based on the knowledge of task priority), worst bottleneck based scheduling algorithms (based on task dependencies) etc.

Please replace the paragraph at lines 1-17 on page 9 with the following:

A thread that requests execution of a specific program is allocated a special-purpose processor 104 ~~202~~ from a pool of same-class special-purpose processors 204, at step 306. The thread may itself be running on a general-purpose processor and request execution of a specific program. Such a thread would temporarily switch from the general-purpose processor mode to the special-purpose processor mode. Once the requested program has been executed, the thread may switch back to the general-purpose processor mode, or request another special-purpose processor. The step of processor allocation is further elaborated upon, with the help of Fig. 4. The thread runs the requested program instance on the processor allocated to it. After complete execution of the program, the thread relinquishes the processor back to processor allocation service 110. At step 308, as soon as the thread releases the control of special-purpose processor 104, it is allocated to one of the other pending threads in the request-queue. This allocation is done in a manner that maximizes the processing efficiency of the multiprocessor and is explained in detail with reference to Fig. 5. Finally, at step 310, the processor goes into idle mode after the request-queue has been exhausted. The exhaustion of the request-queue implies that there are no more pending requests at that moment.